Network Working Group
Request for Comments: 430
NIC: 13299

R. Braden
CCN/UCLA

7 February 1973

COMMENTS ON FILE TRANSFER PROTOCOL

On January 23, 1973, Jon Postel (NMC), Eric Harslem (RAND), Stephen Wolfe (CCN), and Robert Braden (CCN), held and informal meeting at UCLA on FTP. This RFC generally reports the consensus of that meeting on the following issues: server-server transfers (ref. RFC 438 by Thomas and Clements); site-dependent information; and miscellaneous questions/disagreements with RFC 354, 385, and 414. There was also a discussion of the print file muddle, but that subject is addressed in a separate RFC, No. 448.

Miscellaneous Comments on FTP

1. RFC 385, P. 1 (3)

   The question of print files will be discussed at length in another RFC. However, we did feel that the word "still" on the second line from the bottom of Page 1 is gratuitous.

2. RFC 385, P. 2 (5.)
   RFC 385, P. 3 (8.)
   RFC 414, P. 4 (11.i)

   To the extent that we understand these items, they seem to be unnecessary and probably undesirable concessions to particular bad implementations ("hacks"). In reference to the second item, No. 8 in RFC 385, one should note that in an asynchronous multi-process system like the ARPA Network, the phrase "immediately after" has little meaning. An implementation which depends upon "immediately after" is erroneous and should be fixed. If the protocol as defined has an intrinsic race condition, of course, the protocol should be fixed, but we don’t believe such a problem exists. It would help if definitions of command-response sequences in the FTP document were tightened up considerably. As for the last item, we don’t understand why Wayne Hathaway is so strongly opposed to "implied eor".

3. RFC 354, P. 13, Format Definitions for Block Mode

   (a) The definition of the header length presumably is meant to be the "smallest integral number of bytes whose length is greater or equal to 24 bits".

(b) The same definitional problem occurs for restart markers.

(c) Why does the restart marker have to be greater than 8 bits?

(d) Note that changing the Descriptor coding to bit flags would abolish the implied eor as well as the problem of RFC 385, P. 2, #6.

4. RFC 414, P. 5 (11.iii)

Note that text mode is not possible for any EBCDIC coded file. Since EBCDIC is an 8-bit code, Telnet control characters (128-255) cannot be used to distinguish either eor or eof. Stream and block modes will work, however. We have found the diagram on the last page to be useful for keeping track of the three-dimensional space of FTP parameters.

5. RFC 354, P. 17, PASS Command

There is no mechanism within FTP for changing a password. A user shouldn't have to use a different protocol (e.g., log into a time sharing system) to merely change his password.

6. RFC 385, P. 3 (9.), TYPE Before BYTE

This admonition (to send TYPE before BYTE) should be clearly labeled as a recommended procedure for user FTP, not a restriction on a server FTP.

7. RFC 385, P. 2-3 (7) Order of 255 Reply

Some of the participants felt (strongly) that the timing problem dealt with in this item is the result of bad NCP implementations and ought not be dignified in the protocol. The issue here is the old, familiar, and touchy one of queueing RFC's or not. (My own view is that the protocol asymmetry forced by NCP's which can't queue RFC's is at least unaesthetic, and makes some elegant solutions impossible. For examples, see RFC 414 and the comments below on server-server interaction, and RFC 438 on Reconnection Protocol).

8. RFC 354, P. 15, Restart

Following a RESTart command, APPend and STORe presumably have identical meanings.

B. FTP Parameter Encoding

RFC 448, which discusses print files, points out that the print file attribute is logically independent of the character code attribute (ASCII vs. EBCDIC) in the type dimension; the set of allowable types in FTP is the outer product of the individual attributes. Thus FTP has (at least) four character types, summarized by the following two x two matrix:

	ASCII	EBCDIC
Not Print File		
Print File		

I propose that the encoding in the TYPE command model this interdependence of the types. Instead of using a distinct single ASCII character for each type, we should use multiple ASCII characters---qualifiers, if you wish. For example:

A represents ASCII code
E represents EBCDIC code
P represents print file
I represents image
L represents local byte

Then the legal types according to RFC 385 would be:

A
AP
E
EP
I
L

Note that the attributes under consideration here are type-like; they are not (logically) concerned with the structure or the transmission mode, only the internal encoding of the file.

At present, this would be a trivial change. However, I foresee the file transfer protocol expanding significantly over the next several years as new types are added. Some servers will want to add server-specific type variations, and the NWG will want to add some. How about an APL character set? Or the multiple-overlay 256 character ASCII which has been proposed? Multiple qualifiers (and later perhaps more structure) in the type seems to be the cleanest escape mechanism for future growth.

C. Server-Server Interaction

The FTP changes proposed by Thomas and Clements in RFC 438 are a particular solution to a general problem inherent in the current host-host protocol and higher-level protocols like FTP. There seems to be a need for a secure and simple way for two (server) processes in different hosts to exchange socket names (i.e., 40-bit numbers) so they can subsequently exchange RFC's and establish a connection. Current second-level (host-host) protocol provides exactly one (secure) mechanism by which one host can learn a socket name of a process at another host in order to establish a connection: ICP. The ICP mechanism by itself is not adequate for server-server connection in FTP. Therefore, Thomas and Clements have proposed an extension to the FTP protocol, roughly as follows:

(1) A controller ("user") process at Host A uses ICP to invoke and establish Telnet control connections to two automata ("server") processes at two other hosts. An automaton process invoked in this manner then executes controller commands sent in a standard command language over the Telnet control connection.

(2) The controller process commands each automaton to reserve a suitable data transfer socket and to return the socket name to the controller over the control connection. An automaton presumably negotiates with his own NCP in a host-dependent manner to obtain the socket reservation.

(3) The controller now knows both data transfer socket names; he will send them in subsequent commands to the automata so each automaton will know the foreign socket name to which he is to connect. Later commands cause the automata to issue RFC's and open the data connection as needed.

This appears to be useful general model for process-process interaction over the Network. Personally, I believe this symmetrical model should be the basis of all FTP the controller and one of the automata could be in the same host. Then the user/server problem (for any pair of hosts to transfer files, one must have a server FTP and the other a user FTP) would vanish. At least one host somewhere in the Network would need a controller process; all other hosts would need only an automaton process.

Perhaps at a future time the NWG should consider whether a socket-reservation-and-passing mechanism ought to be incorporated into second-level protocol rather than duplicated in a number of third-level protocols. We should note that this model provides secure

sockets only if both user and server processes "release" the socket reservations when the Telnet control connection breaks. The same problem seems to occur with Thomas' Reconnection Protocol (426).

In any case, for the present we would endorse the general third-level model of RFC 438. However, we would propose a slightly different, and more symmetrical, approach.

1. The requirement in FTP that the FTP user listen on the data socket before issuing a data transfer command should be removed. The beauty of host-host protocol is that it doesn't matter which host sends the first RFC, as long as they both send matching RFC's "eventually". (Timeouts, of course, are annoying, but I believe they are workable and ultimately unavoidable); queueing RFC's is also necessary).

2. We propose, instead of LSTN, a new command GETSocket. The controller (i.e., user FTP) process would send a GETSocket to each automaton, probably after a successful login. Upon receiving GETSocket, an automaton would assign a (send, receive) pair of data transfer sockets and return the numbers over the Telnet connection. (Alternatively, FTP could specify that a (send, receive) pair of sockets always be assigned when the server is first entered, and the numbers returned to the user process via unsolicited 255 replies).

3. Then the user process would send the socket numbers to the opposite hosts by sending SOCK commands to both.

4. When it receives a data transfer command, the automaton (server) process would issue an RFC containing the two socket numbers. When both servers are fired up, RFC's are exchanged and data transfer starts.

D. Site-Dependent FTP Parameters

Some hosts will have a problem with the current FTP because their file system needs additional host-specific parameters in certain cases. As an example, the IBM operating systems tend to give the programmer a number of options on the logical and physical mapping of a file onto the disk.

This is true both of TSS/360 (see Wayne Hathaway's discussion of his STOR command implementation, Page 5 of RFC 418), and OS/360. The large set of options and parameters to the OS/360 file system is, in fact, the (legitimate) origin of most complaints about OS Job Control Language (JCL).

If the FTP user merely wants to store data without using it at one of these sites, he has no problem; defaults can be chosen to handle any reasonable FTP request. However, the FTP user who sends a file to an IBM/360 for use there may need to specify local file system parameters which are not derivable from any of the existing FTP commands.

In designing an FTP server implementation for CCN, for example, we first tried to handle the mapping problem by choosing a (possibly different) default mapping for each combination of FTP parameters--type, mode, and structure. We hoped that if a user chose "reasonable" or "suitable" FTP parameters for a particular case (e.g., "ASCII, stream, record" for source programs, and "image, block, record" for load modules), then the right OS/360 file mapping would result. We were forced to abandon this approach, however, because of the following arguments:

1. Some user FTP's probably may not implement all FTP type/mode/structure combinations (though they ought to!).

2. Some user FTP's may not give the user full or convenient control over his type/mode/structure. Indeed, the mode should be chosen on grounds of efficiency, not end use.

3. There weren't enough logically distinct combinations of FTP parameters.

4. The result would have been a set of hard-to-remember rules for sending files to CCN for use here.

5. Some common cases require non-invertible transformations on the data. For example, most IBM language processors (i.e., compilers) accept only fixed length records of (surprise!) 80 bytes each, i.e., literal card images. Such ugly (and logically unnecessary) implementation stupidities in OS/360 are a fact of life. Now if a FTP user innocently sent a data file to CCN with the particular type/mode combination which defaulted to card images, he would find his records truncated to 80 bytes. That would be downright unfriendly.

Thus, the CCN server FTP would have to choose between being useful or being friendly. We decided upon the following strategy:

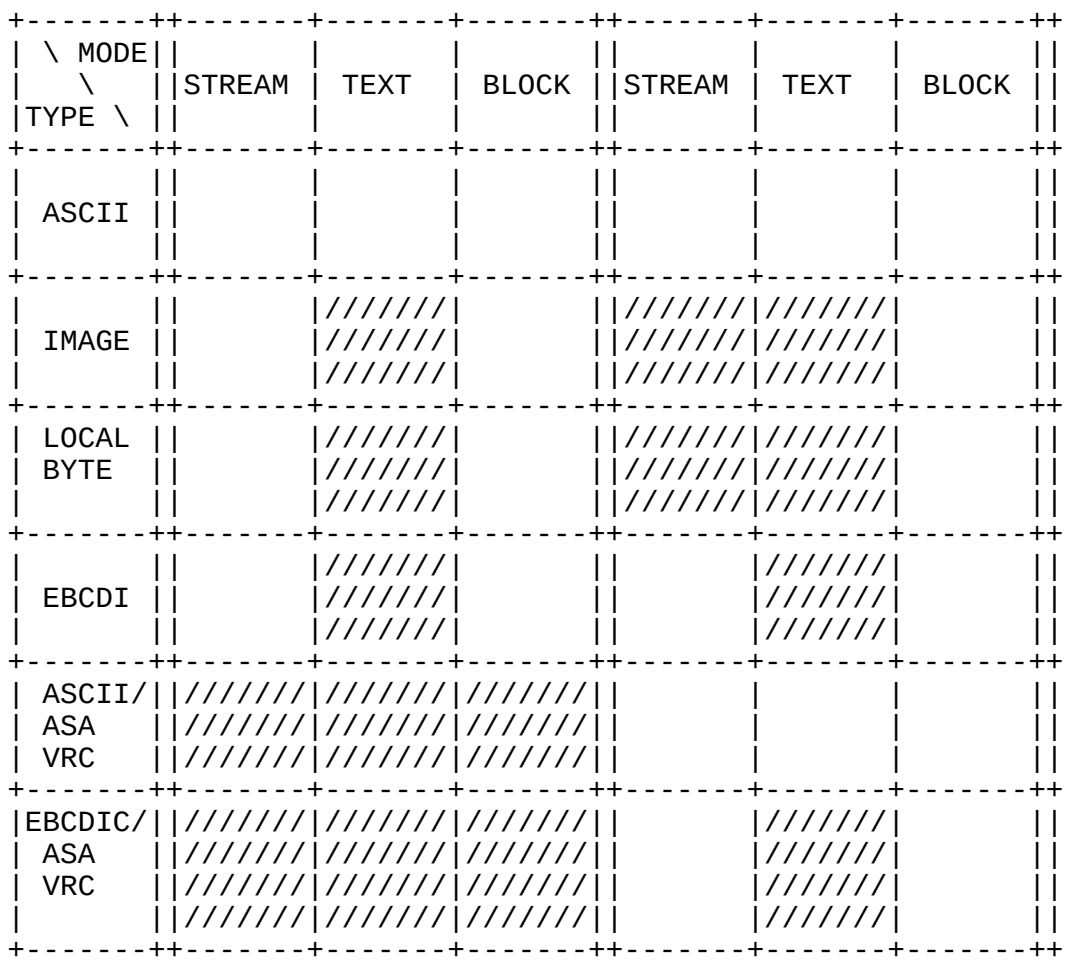
1. The defaults will be friendly; we will accept any FTP type/mode/structure and store it invertibly (except print files). However, the user who uses only these defaults will probably find he has to later run a utility under TSO to reformat the data.

2. We will provide some mnmonic keywords associated with STOR commands to choose the proper disk mapping. For example, if he wants to STORe a Fortran source file for compilation at CCN, the user will need only to specify "SOURCE" or "FORT" to get reasonable and workable OS/360 file system parameters. In addition, we will provide fairly complete "DD" parameters for the sophisticated user. The syntax and semantics of these keywords and parameters will be as close as possible to the corresponding TSO commands. Full details will be published as soon as the implementation is working.

All of this discussion leads to a general protocol question: how should such host-dependent information appear within FTP? Hathaway used the ALLO command (see RFC 418, P. 6). CCN, on the other hand, feels that such information belongs in the only part of FTP syntax which is already host-dependent: the pathname. So CCN plans to allow a "generalized" pathname in a STOR command, a (full or partial) file name optionally followed by one or keywords or keyword parameters separated by commas.

A third possible solution might be for the user to precede his STORe command by a server-dependent data set creation command, using Hathaway's proposed SRVR command. The data set creation command could then have all the parameters necessary for the server file system. CCN might change to this approach if SRVR is adopted and if people find the generalized pathname objectionable or unworkable.

For another interesting example of host-dependent problems, see Hathaway's discussion of his DELE command in RFC 418 (pp.6-7).



[This RFC was put into machine readable form for entry]
[into the online RFC archives by Helene Morin, Via Genie, 12/99]